

大语言模型长文本推断优化技术综述

陶伟^{1,2}, 王健宗¹, 张旭龙¹, 瞿晓阳¹

1. 平安科技(深圳)有限公司, 广东 深圳 518063;

2. 华中科技大学, 湖北 武汉 430070

摘要

随着大语言模型技术的快速发展, 其处理长文本输入的需求日益增加, 但长文本推断面临内存消耗大和时延高的问题。为提高大语言模型在长文本推断中的效率, 对现有优化技术进行了全面回顾和分析。首先, 揭示了影响效率的三大关键因素: 一是庞大的模型体量, 二是具有二次计算复杂度的注意力机制操作, 三是自回归式的解码策略。这些因素共同制约了模型的整体性能表现。随后, 提出了一种分类方法, 将优化技术分为模型优化、计算优化和系统优化, 并详细介绍了量化、稀疏注意力、算子融合等关键技术。研究表明, 这些优化技术能有效提升长文本推断的性能。最后, 展望了未来研究方向, 强调了进一步优化大语言模型长文本推断的重要性, 以满足不断增长的长文本推断需求。

关键词

大语言模型; 长文本推断; 模型优化; 计算优化; 系统优化

中图分类号: TP391

文献标志码: A

doi: 10.11959/j.issn.2096-0271.2025067

Long-context inference optimization for large language models: a survey

TAO Wei^{1,2}, WANG Jianzong¹, ZHANG Xulong¹, QU Xiaoyang¹

1. Ping An Technology (Shenzhen) Co., Ltd., Shenzhen 518063, China

2. Huazhong University of Science and Technology, Wuhan 430070, China

Abstract

With the rapid development of large language model (LLM) technology, the demand for processing long-text inputs has been increasing. However, long-text inference faces challenges such as high memory consumption and latency. To improve the efficiency of LLMs in long-text inference, a comprehensive review and analysis of existing optimization techniques were conducted. The study first revealed three key factors that affect efficiency: the first is the huge model size, the second is the attention mechanism operation with quadratic computational complexity, and the third is the autoregressive decoding strategy. These factors together restrict the overall performance of the model. Subsequently, a taxonomy was proposed, categorizing optimization techniques into model optimization, computation optimization, and system optimization, with detailed introductions to key technologies such as quantization, sparse attention, and operator fusion. The research results demonstrate that these optimization techniques can effectively enhance the performance of long-text inference. Finally, future research directions were outlined, emphasizing the importance of

further optimizing LLMs for long-text inference to meet the growing demands of context length.

Key words

large language model, long-context inference, model optimization, computation optimization, system optimization

0 引言

大语言模型（large language model, LLM）是一类基于深度学习的模型，专门用于处理和生成自然语言文本。其核心架构通常基于变换器（transformer）^[1]，通过大量语料的预训练来学习语言中的语义和上下文关系。与传统的自然语言处理模型不同，LLM能够高效地理解复杂句子的结构，进行文本生成、翻译、摘要和对话等多种任务。代表性的LLM包括来自变换器的双向编码表示（bidirectional encoder representations from transformers, BERT）^[2]、生成式预训练变换器（generative pre-trained transformer, GPT）^[3-6]系列等，前者专注于语言理解，后者侧重于生成任务。

LLM的应用主要分为长文本建模与长文本推断。长文本建模目标是让模型有效地处理并理解长文本中的信息，并通过学习长文本中的语境来生成准确的语言表示，常用于文本摘要或者情感分析等任务。传统的方法包括基于循环神经网络（recurrent neural network, RNN）、基于Transformer的模型（如BERT、GPT）或者改进的Transformer模型（如Longformer、Reformer等）。长文本建模的主要解决思路在于采用分段处理或窗口滑动等技术，将长文本分成多个较短的片段，逐段进行建模，之后再通过聚合策略将信息进行整合。同时，现代的长文本建

模也尝试通过注意力机制来选择性地关注重要信息，从而避免对冗长、无关信息的处理。长文本推断指的是通过长文本中的信息推理出隐含的结论或解答，通常包括逻辑推理、常识推理或从多条信息中进行归纳推理。例如，问题回答系统、情境推理、阅读理解等任务都是推断任务，模型需要从长文本中提取关键线索，推导出合适的答案。长文本推断常见的方法包括基于图神经网络（graph neural network, GNN）方法建模文本中的实体关系、基于Transformer的阅读理解模型、检索增强生成（retrieval-augmented generation, RAG）等。其主要解决思路是通过构建多层推理模型来逐步获取信息，例如通过多轮的推理过程来验证假设或通过外部知识库来补充推理信息。两者都需要高效的上下文建模，且都广泛用到Transformer及其变种。不同之处在于它们的目标：长文本建模任务侧重于文本的整体理解与表示；长文本推断任务侧重于从文本中进行逻辑推理，得出结论或答案。

随着模型参数规模的迅速增长，LLM长文本推断展现出强大的多任务适应性和跨领域应用能力，被广泛应用于搜索引擎、智能助手、内容创作等领域。随着LLM长文本推断的应用范围越来越广泛，人们对LLM上下文长度的需求也越来越迫切。因为上下文长度的增加能带来很多优势，包括：能处理更长的文本段落，提高总结、翻译、代码解析等任务的连贯性，同时减少上下文切换时的信息丢失，允许模型在处理长文档、复杂对话或跨章节分析时保

持一致的语义理解和引用信息。研究者提出了很多方法来扩展 LLM 的上下文，其成果也十分显著。近年来，LLM 的上下文长度从早期的数百个标记（token）增加到如今的上万甚至 10 万个标记。例如，OpenAI 的 GPT-4 Turbo^[6] 支持 128 000 个标记的上下文长度。然而，过长的上下文也会给 LLM 的推断带来很多挑战，如产生更高的计算和存储成本、可能导致推断速度减慢，并且使 LLM 无法部署在资源受限的设备上。

为了全面了解现有的研究并激发进一步的研究，本文系统地总结了 LLM 推断的现状与挑战。首先，介绍了 LLM 的基本概念和知识，并详细分析了 LLM 在长文本推断过程中的效率瓶颈；然后，将 LLM 长文本推断优化技术分为 3 个层次（模型优化、计算优化、系统优化），并从这 3 个方向分别讨论了 LLM 长文本推断优化的技术；最后，展望 LLM 长文本推断优化技术未来的研究方向。

本文的主要贡献如下。

- 介绍了 LLM 的背景知识，分析了 LLM 在长文本推断过程中存在的显存消耗与推断时延问题。
- 将 LLM 长文本推断优化技术分为模型优化、计算优化、系统优化 3 类，并分别讨论了现有的 LLM 长文本推断优化技术。
- 对 LLM 长文本推断优化技术未来的研究工作进行展望，强调了几个需要特别关注的研究方向。

1 大语言模型长文本推断概述

LLM 的出现最早源于以 Transformer 为基础的模型架构，其使 LLM 通过自注意

力机制理解文本中的长距离依赖，从而具备更高效的文本理解和生成能力。通过大量的参数与语料的训练，LLM 在多个领域取得显著成果，例如文本生成、翻译、摘要和对话。近年来，随着模型能力的增强，LLM 开始被应用于处理更大规模的文本，如书籍、长篇文章和法律文档等。然而，当被应用于长文本推断时，LLM 面临很多挑战，如计算时间、内存消耗等。如何解决 LLM 长文本推断问题成为现在 LLM 研究中的一个核心任务。

本节将综合介绍大语言模型长文本推断的背景知识，首先回顾大语言模型出现的背景知识，包括 Transformer 是如何形成的、Transformer 的计算原理、Transformer 结构的特点、现代 LLM 的发展状况等。然后，深入分析大语言模型在执行长文本推断时的挑战，包括 LLM 的推断过程、键值缓存（key value cache, KV Cache）的设计、长文本情况下可能出现的挑战等。

1.1 大语言模型背景知识

主流 LLM 都是基于 Transformer 框架构建的。Transformer 模型的提出源于自然语言处理（natural language processing, NLP）领域对更高效、更精确模型的迫切需求。此前，RNN^[7] 和 LSTM^[8] 虽然在处理序列数据上有一定效果，但它们在捕捉长距离依赖时存在困难，而且由于序列处理的顺序性，训练速度较慢，无法有效并行化。为了解决这些问题，Google 于 2017 年提出了 Transformer 模型^[1]，它完全基于注意力机制，能够并行处理序列数据，并通过自注意力机制捕捉句子中不同词之间的关系，从而显著提升了 NLP 任务的性能和效率，推动了机器翻

译、文本生成等领域的发展。

如图1所示,标准的Transformer架构由多个依次堆叠的Transformer模块构成。每个Transformer模块通常包含3个核心组件:多头自注意力机制(multi-head self-attention, MHSA)模块、前馈神经网络(feed forward network, FFN)模块以及层归一化(layer normalization, LN)模块。这些模块以级联方式工作,每个模块接收前一模块的输出特征作为输入,并通过内部的子模块对特征进行处理和转换。在数据输入阶段,系统首先通过标记化处理将原始输入文本转换为标记序列。随后,嵌入层将这些标记映射为具有语义信息的特征向量。为了保留序列的顺序信息,系统还会在输入特征中加入位置编码(position embedding),从而为每个标记的位置信息进行编码。这种结构设计使Transformer能够有效处理序列数据,同时保持对元素位置关系的敏感性。

Transformer体系结构的核心概念是在MHSA模块中采用自注意力机制。假设输入的token的词向量为 $\mathbf{X}=[x_1, x_2, \dots, x_n]$, MHSA模块会分别使用查询权重矩阵 $\mathbf{Q}^w$ 、键权重矩阵 $\mathbf{K}^w$ 、值权重矩阵 $\mathbf{V}^w$ 对其进行线性投影,得到查询矩阵 $\mathbf{Q}$ 、键矩阵 $\mathbf{K}$ 和值矩阵 $\mathbf{V}$,对于每个注意力头都会有这样的计算,如式(1)所示:

$$\mathbf{Q}_i = \mathbf{XW}^{\mathbf{Q}_i}, \mathbf{K}_i = \mathbf{XW}^{\mathbf{K}_i}, \mathbf{V}_i = \mathbf{XW}^{\mathbf{V}_i} \quad (1)$$

其中, $\mathbf{W}^{\mathbf{Q}_i}$ 、 $\mathbf{W}^{\mathbf{K}_i}$ 、 $\mathbf{W}^{\mathbf{V}_i}$ 为第*i*个注意力头对应的投影矩阵。然后对每个 $(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$ 矩阵元组进行自注意力运算,其中先对查询矩阵 $\mathbf{Q}_i$ 和键矩阵 $\mathbf{K}_i$ 进行计算,得到注意力权重矩阵 $\mathbf{A}_i$,如式(2)所示:

$$\mathbf{A}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i) = \frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}} \quad (2)$$

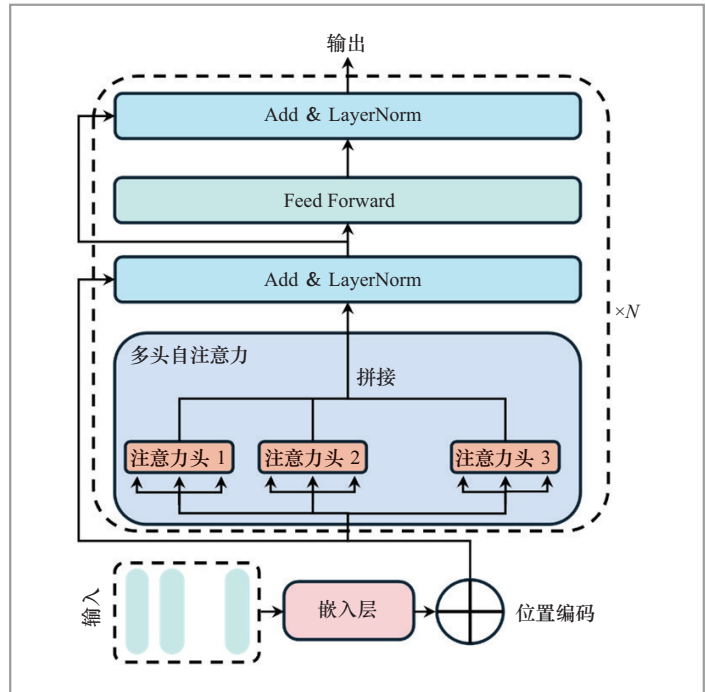


图1 Transformer结构示意图

其中, d_k 是token词向量的维度。计算得到的注意力权重表示当前token与整个token序列中其他token的关系,也可以视为整个token序列中其他token对当前token的影响。然后对注意力权重进行处理并与值矩阵 $\mathbf{V}_i$ 相乘,得到这一个注意力头的特征 $\mathbf{Z}_i$,如式(3)所示:

$$\mathbf{Z}_i = \text{Softmax}(\mathbf{A}_i + \mathbf{M}) \mathbf{V}_i \quad (3)$$

其中, $\mathbf{M}$ 矩阵是掩码矩阵,作用是确保token序列中输入token之后的token的注意力权重被置为0。这是出于输入序列的因果性考虑的,因为对于输入token之后的token,从输入token的角度看,属于未来的token,未来的token当然不可能对现在的token产生影响,所以其注意力权重也要被置为0。使用Softmax函数的目的是保证其他token对当前token的影响之和为1。注意力计算由于涉及矩阵乘法,

其时间复杂度和空间复杂度是输入 token 序列长度的二次方。最终，MHSA 模块将所有注意力头的特征进行拼接，并通过线性变换层进行处理，从而生成输出特征 Z ，这一过程如式 (4) 所示：

$$Z = \text{Concat}(Z_1, Z_2, \dots, Z_h)W^o \quad (4)$$

其中， W^o 是投影矩阵。自注意力机制的优势在于能够突破距离限制，动态评估输入序列中各部分的重要性权重。这种特性使其能够有效捕捉输入文本中的长距离依赖和复杂语义关联。

在 Transformer 架构中，另一个关键组件是 FFN。该模块通常接续在 MHSA 模块之后，其由两个线性变换层构成，并在层间引入非线性激活函数以实现更强大的特征表达能力。它从 MHSA 模块接收输出特征 X ，按照式 (5) 进行处理：

$$\text{FFN}(X) = W_2 \sigma(W_1 X) \quad (5)$$

其中， W_1 和 W_2 表示两个线性变换层的权重， $\sigma(\cdot)$ 表示激活函数。

基于 Transformer 架构，研究者开始训练规模更大、数据更多的模型，逐渐演变为如今的 LLM。随着计算资源的提升和数据的积累，LLM 通过预训练和微调技术，能够生成、理解复杂文本并进行推理。

目前，热门的 LLM 包括 GPT 系列（如 GPT-4^[6]）、BERT 及其改进版本（如 RoBERTa^[9]），以及 Google 的 PaLM^[10] 和 OpenAI 的 Claude。这些模型广泛应用于文本生成、机器翻译、问答系统等领域，推动了 NLP 的快速发展。

1.2 大语言模型推断过程

在 LLM 中，最容易发现的消耗资源的组件是 FFN 模块。FFN 模块的参数数量特别大，例如，FFN 模块参数量在 LLaMA-7B^[11] 模型中占 63.01%，在 LLaMA-70B 模型中占 71.69%。因此，其在推断时也会消耗非常多的显存，并花费较长的时间进行计算。此外，在长文本推断中，因为当前 LLM 的架构特性，还存在其他问题。

图 2 展示了当前流行的 LLM 的推断过程，其中菱形框内为每个输出的 token。当前流行的 LLM 都是仅解码器 (decoder-only) 架构，这类架构通常通过自回归机制逐步生成文本序列。具体而言，仅解码器架构的 LLM 的推理过程可划分为以下两个主要阶段。

- 预取阶段 (prefill stage)：在这个阶段中，LLM 接收整个输入 token 序列，处理后得到输出序列的第一个 token。

- 解码阶段 (decoding stage)：在这个阶段中，LLM 将整个输入 token 序列和已经得到的输出 token 序列结合起来处理，得到输出序列中的下一个 token，这个过程将迭代进行，直到输出句子的结束符 (<End of Sentence>, <EOS>)。

上述 LLM 处理过程中主要的计算量集中在前文提到的多头自注意力计算环节。由于解码阶段是迭代进行的，并且无法并行执行，由此引出了大模型推断过程中的第一个问题：每个解码阶段都需要计算整

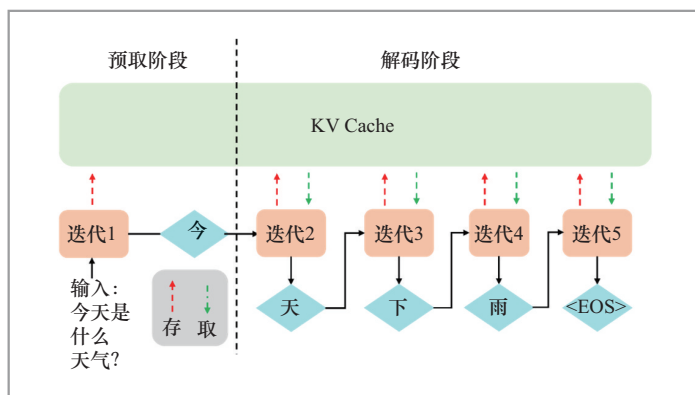


图2 LLM推断流程

个输入 token 序列和已经得到的输出 token 序列的键矩阵 $\mathbf{K}$ 和值矩阵 $\mathbf{V}$ ，这会带来大量的冗余计算，显著提高推断时延。为此，研究者设计出了 KV Cache。KV Cache 将输入 token 序列和已经得到的输出 token 序列的键矩阵和值矩阵都缓存在图形处理单元（graphics processing unit, GPU）显存中，当需要计算时再取出调入高速缓存（flash）中。当计算出一个新的输出 token 的 $\mathbf{K}$ 矩阵和 $\mathbf{V}$ 矩阵以后，再将其同样缓存起来，附加在已有的 KV Cache 后面。

但是，KV Cache 的引入，在 LLM 长文本推断时会带来很大的挑战。引入 KV Cache 以后，LLM 推断时的显存消耗与推断时延都与 KV Cache 的大小成正比。在长文本推断中，输入的序列很长，导致 KV Cache 也很大，如此大的 KV Cache 会带来很高的显存占用，而且常常比 LLM 本身的参数占用的显存还要高。例如，在 Facebook 公司发布的 Llama2 13B^[12] 模型中，一个长度为 128 000 个标记的序列需要 100 GB 显存。与之相比，目前业界最流行的科研用 GPU——Nvidia A100 的显存只有 80 GB。此外，过大的 KV Cache 每个解码阶段都要从内存中调入 GPU 显存中，这个过程非常耗时，且会消耗 GPU 的带宽，导致整个推断过程的时延变得很高。

因此，为了解决长文本推断中的显存占用、推断时延等挑战，研究者探索了各种优化技术，以更好地应对这些挑战。

2 大语言模型长文本推断优化技术

随着 LLM 的快速发展，其在自然语言处理中的应用范围越来越广泛，特别是在处理长文本推断任务时表现出强大的生成能力。然而，随着模型规模的不断扩大，

推断过程中的计算和内存消耗问题也日益凸显，这不仅限制了 LLM 在资源受限环境中的部署，还提高了推断的时延和成本。因此，如何提升长文本推断效率，成为亟待解决的关键问题。近年来，研究人员提出了多种优化方法，包括模型优化、计算优化与系统优化，极大提升了推断效率和适应性。

本章将系统综述近年来在 LLM 推断优化领域的主要研究进展。本文系统探讨了提升长文本推理效率的 3 类优化方法：① 模型优化，改进模型架构（如高效注意力机制）和训练方式（如知识蒸馏、量化），在减少参数量的同时保持精度；② 计算优化，通过并行计算策略（张量/流水线并行）、算子融合等技术提升计算效率；③ 系统优化，结合软硬件协同设计（如分页注意力、批处理优化）改善资源利用率。这些方法各具特点又相互补充，共同推动了长文本处理在效率和质量上的突破，为实际应用提供了重要参考。

2.1 模型优化

LLM 长文本推断的模型优化主要集中在 3 个方面：设计高效注意力机制、设计高效模型架构以及模型压缩。设计高效注意力机制包括共享 attention、稀疏 attention、近似 attention。设计高效模型架构主要涉及混合专家（mix of experts, MoE）、自动搜索模型架构以及设计 Transformer 架构替代品。模型压缩则主要包括量化技术、知识蒸馏等。

2.1.1 设计高效注意力机制

各种设计高效注意力机制的技术及其对比见表 1。

共享 attention 的核心思想是共享 LLM

表 1 不同设计高效注意力机制技术比较

分类	技术	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
共享 attention	MQA ^[13]	2019 年	中	高	中	低	中	中	中
	GQA ^[14]	2023 年	中	较高	较高	中	较高	较高	较高
	MiniCache ^[15]	2024 年	较高	高	较高	中	较高	较高	较高
	LCKV ^[16]	2024 年	高	较高	中	低	较高	较高	高
	CLA ^[17]	2024 年	较高	高	较高	中	较高	较高	高
稀疏 attention	StreamingLLM ^[18]	2023 年	中	较高	中	低	中	中	中
	Longformer ^[19]	2020 年	中	高	较高	低	高	高	高
	H ₂ O ^[20]	2024 年	高	较高	高	较低	高	高	高
	Infini-Transformer ^[21]	2024 年	中	高	高	高	较高	较高	较高
近似 attention	Performer ^[22]	2020 年	中	较高	中	低	较高	较高	较高
	LinFormer ^[23]	2020 年	较高	高	较高	低	高	高	高
	LongLoRA ^[25]	2023 年	中	高	高	中	高	高	高
	CaraServe ^[26]	2024 年	较高	高	高	中	高	高	高

中 attention 计算的结果，从而避免大部分复杂耗时的 attention 计算，并节省存储这些计算结果所需的显存。该技术最早起源于多查询注意力（multi-query attention，MQA）^[13] 论文。LLM 在进行自注意力计算时，需要计算多个注意力头的键矩阵 $\mathbf{K}$ 和值矩阵 $\mathbf{V}$ ，而 MQA 让 LLM 的多个注意力头共享 KV Cache，这样做能显著减少显存消耗和显存访问时间，缺点是共享 KV Cache 会降低模型的精度。随后组查询注意力（group-query attention，GQA）^[14] 论文进一步优化了这一技术，它提出一种折中的方案，将 LLM 的所有注意力头分为若干分组，每个组内共享 KV Cache，而组间的 KV Cache 仍不同。这样能减轻共享 KV Cache 对模型精度的破坏。除了在同层内共享 attention，研究者还提出了在不同层之间共享 attention 的方法。文献[15]提出在相邻的两层之间共享 KV Cache，为了降低损失，它将 KV Cache 的向量分为幅度分量和方

向分量，保持幅度分量不变，只合并方向分量。对于那些高度不一致的向量，保留它们，不合并。LCKV^[16] 只计算 Transformer 顶层的 KV，并共享给其他所有层，将所有层的 query 与之配对。CLA^[17] 则将 LLM 的所有层分为若干组，每一组内共享 KV Cache。

稀疏 attention 的核心思想是选择性关注，即只对输入序列中的一部分位置进行注意力计算，而不是全部，其余位置的 attention 结果则置 0 处理。StreamingLLM^[18] 通过实验证实初始位置的 token 更加重要，为此作者在滑动窗口 attention 的基础上额外保留了 KV Cache 中初始位置的 token。Longformer^[19] 采用了滑动窗口 attention 和全局 attention 的结合。滑动窗口 attention 只对每个 token 的局部邻域进行计算，而全局 attention 则允许模型在某些关键位置（如序列的开头或特定标记）上保留全局信息。H₂O^[20] 通过计算每个 token 的注意力权重之和来确

定 token 的重要性，并定期从 KV Cache 中剔除掉 attention weight 最低的 token，以维持总的 token 数量不超过预设的上限。Infini-Transformer^[21]采用当前输入与部分历史输入融合的方法，不同之处在于它缓存历史记忆时只存储了历史记忆的压缩结果（只保存了 $\mathbf{K}-\mathbf{V}$ 矩阵之间的绑定关系和归一化项）。

近似 attention 技术则使用各种简单的计算方法来代替自注意力计算中的矩阵乘法与 Softmax 函数等复杂的计算操作，从而降低计算的时间复杂度和空间复杂度。例如，Performer^[22]将 Softmax 核函数分解为随机特征映射，然后用近似的方式处理 Softmax 的相似度计算。该方式不再需要计算所有输入的配对关系，而是通过一组正交随机特征来对输入进行投影，并将这些投影后的值进行加权求和。这种方法能够将两次复杂度的 Softmax 函数近似为线性的矩阵投影。LinFormer^[23]引入低秩分解（low rank adaption, LoRA）^[24]，低秩分解的计算式见式（6）：

$$\mathbf{A} \approx \mathbf{P}\mathbf{Q} \quad (6)$$

其中， $\mathbf{A}$ 矩阵大小为 $m \times n$ ， $\mathbf{P}$ 矩阵大小为 $m \times k$ ， $\mathbf{Q}$ 矩阵大小为 $k \times n$ ， k 是 $\mathbf{A}$ 矩阵的秩且 k 远远小于 m 和 n 。这种方式可以将 $\mathbf{A}$ 矩阵近似为两个参数量远远小于 $\mathbf{A}$ 的矩阵的乘积。LinFormer 对自注意力计算中键矩阵 $\mathbf{K}$ 和值矩阵 $\mathbf{V}$ 应用低秩分解，大大减少键矩阵和值矩阵的参数量，从而减少内存消耗，并且更少的参数量意味着更少的计算量，也能大大减少推断时间。LongLoRA^[25]则是将低秩分解操作进一步扩展到了嵌入层和归一化层，实验证明这样做能更好地适应长文本推断。CaraServe^[26]在 GPU 上维护基础模型，并从主内存动态加载激活的 LoRA 适配器。同时它采用 CPU 辅助方法，在将激活的适

配器加载到 GPU 上时，会在 CPU 上尽早启动它们进行预填充；加载完成后，再切换到 GPU 生成 LoRA 推理。

2.1.2 设计高效模型架构

各种不同设计高效模型架构技术对比见表 2。

混合专家（MoE）^[27]技术的核心思想是将模型分为多个“专家”（子网络），在推理时只激活其中一部分专家，基于输入动态选择，在减少计算成本的同时保持模型性能。MoE 技术使用路由机制将输入的 token 分配给特定的专家，从而确保网络中仅有相关部分参与计算。这种选择性激活显著降低了浮点运算（floating point operations, FLOPs），同时保留了模型的高效表现。将 MoE 用于 LLM 中 FFN 的做法最早源于 Google 提出的 Switch Transformer^[28]，它将每个并行计算的 FFN 视为 MoE 中的专家。Google 随后又提出了 GLaM^[29]，它是 Switch Transformer 的改进版本，不同之处在于 Switch Transformer 中一个 token 只能激活一个专家，而 GLaM 中一个 token 可以激活多个专家。MoEfication 论文^[30]提出了一种将现有的非 MoE 模型转化为 MoE 架构的方法，而无须从零开始训练。该方法通过对模型的 FFN 进行专家分割，将 FFN 神经元分组为不同的专家。BASE 论文^[31]提出了一种优化 MoE 模型进行专家分配的新方法。其核心是通过拍卖算法解决专家与输入 token 的分配问题，确保每个专家处理的 token 数量平衡，以避免一些专家过载而其他专家空闲的情况出现。FactorLLM^[32]先将训练后的密集 LLM 使用因式分解的方法分解为稀疏子网络，然后应用 MoE 方法，同时在 MoE 结构中动态激活部分专家，以减少推理时的计算负担。

表2 不同设计高效模型架构技术对比

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
混合专家	SwitchTransformer ^[28]	2022年	中	高	高	低	高	中	高
	GLaM ^[29]	2022年	较高	高	较高	中	高	较高	较高
	MoEfication ^[30]	2021年	中	中	中	低	中	中	中
	BASE ^[31]	2021年	较高	高	高	中	较高	较高	较高
	FactorLLM ^[32]	2024年	高	高	较高	中	高	高	高
自动搜索模型 架构	AutoTinyBert ^[34]	2021年	高	较高	高	高	高	高	高
	LLaMA-NAS ^[36]	2024年	高	较高	高	高	高	高	高
	StructuralPruning ^[37]	2023年	高	较高	高	高	高	高	高
	LiteTransformerSearch ^[38]	2022年	中	高	较高	中	高	高	高
Transformer 替代架构	S4 ^[39]	2021年	高	高	高	低	高	高	高
	Mamba ^[40]	2023年	高	高	高	较低	高	高	高
	RWKV ^[41]	2023年	高	高	较高	较低	较高	较高	较高
	xLSTM ^[42]	2024年	高	较高	高	高	较高	较高	高
	minGRU&minLSTM ^[43]	2024年	中	高	中	低	中	中	中

自动搜索模型结构主要用到了神经架构搜索（neural architecture search, NAS）技术^[33]。它通过搜索算法，从大量候选架构中搜索最优的神经网络结构、连接方式和参数，而不需要依赖人工设计。NAS技术可以实现模型推断时间、显存消耗以及模型精度的综合最优，即在搜索到最高效 LLM 的同时保证模型的精度不会太差。NAS 技术最早用于传统的深度神经网络，但近年来它也越来越频繁地被用于搜索高效的 LLM 结构。AutoTinyBERT^[34]创新性地采用了 one-shot NAS^[35]（一种基于神经架构搜索的改进方法，其核心原理是在训练过程中保留所有可能的操作路径，而在推理阶段仅选择最优路径）来优化 Transformer 架构的超参数配置。该方法应用了一种创新的批处理训练策略，用于训练超级预训练语言模型（SuperPLM），并在此基础上结合进化算

法来筛选性能最优的子模型架构。LLaMA-NAS^[36]使用一种基于 LLaMA2-7B 使用一次性 NAS 寻找帕累托最优网络架构的有效方法。具体来说，他们只对 LLaMA2-7B 进行一次微调，然后应用基于遗传算法的搜索来寻找更小、计算复杂度更低的网络架构。有些方法将 NAS 技术与模型压缩相结合，用 NAS 技术搜索最优的模型压缩方式。Structural Pruning^[37]将 LLM 剪枝视为一个多目标 NAS 问题，并使用 one-shot NAS 来搜索最优解。此外，NAS 的搜索一般比较耗时，还需要专门训练一个搜索用的网络。因此也有一些工作致力于在将 NAS 应用到搜索 LLM 结构时优化 NAS 本身。文献^[38]提出使用不需要训练的指标，即参数的数量作为代理指标来指导搜索，这种方法可以大大减少搜索所需时间，并且不需要在搜索阶段进行实际训练。

研究者还提出了其他高效 LLM 架构来代替计算复杂的 Transformer 架构。例如，结构化状态空间模型（structured state space model, S4）^[39]基于状态空间模型（state space model, SSM）来建模。其使用结构化的状态空间方程捕捉输入序列的时间动态，将长序列任务的复杂性降低到线性，从而高效处理长距离依赖任务。状态空间模型本质上是递归的，因此它能够高效地对序列数据进行计算，而不需要像 Transformer 那样依赖全局自注意力机制。记忆增强广播注意力（memory-augmented broadcast attention, Mamba）^[40]是一种结合了记忆增强机制的新型注意力模型，其核心思想是引入一个全局的记忆模块来存储和传递长程依赖信息，而不需要对每个输入都进行全局的自注意力计算。这样，Mamba 能够显著降低计算开销，同时保留捕捉长距离依赖的能力。循环加权键值（recurrent weighted key value, RWKV）模型^[41]是一种结合了 RNN 和 Transformer 机制的新型架构，旨在保留 RNN 的记忆能力和 Transformer 的有效性。RWKV 模型将 Transformer 的键值对结构引入 RNN 中，使模型可以像 RNN 一样逐步处理输入序列，同时保留了 Transformer 中捕捉上下文信息的能力。xLSTM^[42]是一种新型 LSTM 架构，旨在将传统神经网络架构 LSTM 扩展到数十亿参数的规模，用它设计了两种 LSTM 变体，分别是 sLSTM 和 mLSTM。sLSTM 引入了指数门控和新的存储混合技术，允许 LSTM 修订其存储决策；mLSTM 将 LSTM 的记忆单元从标量扩展到矩阵，提高了存储容量，并引入了协方差更新规则，使 mLSTM 可以完全并行化。将 sLSTM 和 mLSTM 集成到残差块中，就构建成了 xLSTM 架构。Feng 等^[43]通过从 GRU、

LSTM 的各种门中删除它们以前的状态依赖，以及消除对输出范围的约束的方式，简化了 GRU 和 LSTM，获得参数量更少的且能并行训练的 minGRU 和 minLSTM。

2.1.3 模型压缩

各种模型压缩技术见表 3。

量化^[44]指的是将模型中数据的类型从高精度转变为低精度（例如从 32 位浮点数转为 8 位整型），低精度的数据消耗的内存更少且计算量更小，但是也可能带来模型精度的严重损失。早期的量化主要着眼于模型的权重与激活值。GPTQ^[45]是一种新的训练后量化方式，其核心是基于梯度信息对量化误差进行最小化调整。它通过逐层量化模型参数并结合梯度反馈来校正量化后的参数，这样可以有效减少模型参数的精度损失。LLM.int80^[46]使用张量级量化，将矩阵乘法视为行向量和列向量的独立内积序列，然后对每个内积使用单独的量化归一化常数来提高模型精度。同时，对于那些离群特征维度（特征幅度明显比其他维度大），它保留其精度在 32 位浮点数。由于离群特征维度数量很少但是对模型精度影响很大，将它们的精度保留在 32 位浮点数能够很好地维护模型精度，同时不会带来太多的显存与计算开销。SqueezeLLM^[47]将这些离群特征维度提取出来单独作为一个稀疏矩阵，其余特征维度则作为一个稠密矩阵，然后保留稀疏矩阵的精度，同时量化稠密矩阵到 8 位整型。AWQ^[48]将离群特征维度进行重排序，使排序后非离群特征维度是物理连续的，离群特征维度也是物理连续的，然后进行混合精度量化，非离群特征维度量化为 8 位整型，离群特征维度保留 32 位，重排序能够有效避免混合精度量化时的硬件效率亏损。随着 LLM 技术的发展，KV Cache 对推断

表3 不同模型压缩技术比较

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
量化	GPTQ ^[45]	2022年	中	中	高	中	较高	较高	较高
	LLM.int8 ^[46]	2022年	中	高	较高	低	高	高	高
	SqueezeLLM ^[47]	2023年	较高	高	较高	中	较高	较高	较高
	AWQ ^[48]	2023年	高	较高	较高	较低	高	高	高
	KIVI ^[49]	2024年	高	高	中	低	较高	较高	高
	MiKV ^[50]	2024年	较高	中	较高	中	中	中	较高
	FlexGen ^[51]	2023年	中	高	较高	低	较高	较高	较高
知识蒸馏	MiniLLM ^[53]	2023年	较高	高	较高	较低	较高	较高	较高
	GKD ^[54]	2023年	高	较高	高	中	高	高	高
	MCKD ^[55]	2023年	高	较高	较高	中	较高	较高	较高
	Lamini-LM ^[56]	2023年	中	中	较高	较低	中	中	中
	LLM-Neo ^[57]	2024年	中	高	较高	中	中	高	中

效率的影响越来越明显，越来越多的量化技术也开始着眼于 KV Cache。KIVI^[49]通过实验发现键矩阵中离群特征维度较多而值矩阵中离群特征维度较少，因此它对键矩阵和值矩阵分别使用逐通道量化和逐 token 量化，将 LLM 量化到极低精度的 2 位整型，同时不损失太多模型精度。MiKV^[50]通过计算注意力分数矩阵中每一列的和（与之前提到的 H₂O 类似）的方式寻找相对重要的 token，对于重要 token 保留精度为 32 位浮点数，其余 token 则量化到整型。FlexGen^[51]对 KV Cache 采取组量化，将键矩阵和值矩阵分为若干分组，每个组内的量化参数保持一致，从而减少量化过程的计算量。

知识蒸馏^[52]指的是将原始的较大的 LLM 作为教师模型，然后重新训练一个规模较小的 LLM，将这个小 LLM 作为学生模型去学习大 LLM 的知识，然后用小 LLM 代替大 LLM，这样能够以更高效的小 LLM 来完成大 LLM 的复杂任务。知识蒸馏可分为白盒知识蒸馏与黑盒知识蒸馏。

白盒知识蒸馏指能够访问到教师模型的中间特征和输出逻辑。MiniLLM^[53]在标准白盒知识蒸馏方法的基础上，将前向库尔巴克-莱布勒（Kullback-Leibler, KL）散度替换为更适合生成语言模型的后向 KL 散度，从而提高长文本生成时的质量。GKD^[54]引入了广义知识蒸馏，其不仅仅依赖于一组固定的输出序列，而是利用教师对这些序列的反馈，在其自生成的输出序列上训练学生。与监督式知识蒸馏方法不同，GKD 还提供了学生和教师之间的替代损失函数，当学生缺乏模仿教师分布的表达能力时，这会很有用。黑盒知识蒸馏指教师模型的中间特征和输出逻辑不可访问到的知识蒸馏方法。其仅适用于用教师模型学到的最终结果来训练学生模型，主要目的是引导学生模型学习教师模型的泛化能力。MCKD^[55]提出了多阶段协作知识蒸馏以解决小样本领域的知识蒸馏问题。它首先使用少量样本来提示 LLM 为未标记数据生成伪标签；然后在迭代知识蒸馏过程的每个阶段，在伪标记数据的不相交分区

上训练一对新的学生模型，并在这两个学生模型看不见的分区生成新的和改进的伪标签。Lamini-LM^[56]通过多样化的指令集来提炼学生模型，从而学习教师模型的指令微调能力，这些能力通常是资源需求很高的，使用学生模型能显著降低其资源需求。知识蒸馏和低秩分解有相同的范式，LLM-Neo^[57]将低秩分解集成到知识整理中，以提高知识传递的效率。

2.2 计算优化

LLM 长文本推断的计算优化主要集中在 3 个方面：并行算法、内存访问优化算法以及计算图优化。并行算法主要分为数据并行、流水线并行和张量并行。内存访问优化技术包括分块和预取技术。计算图优化包括算子融合和动态早退。

2.2.1 并行算法

各种不同并行算法的对比见表 4。
数据并行的主要目的是加快 LLM 推断的速度。具体操作是将数据集分成若干个碎片分别放在多个 GPU 上，同时每个 GPU 上放置一份模型副本，然后多个 GPU

同时开始推断。早期的数据并行适用于单机多卡的场景，采用参数服务器(parameter server)^[58]框架，近来又发展出了分布式数据并行技术，即在多机多卡上同时进行推断，采用 Ring AllReduce^[59]的通信方式。

流水线并行指的是将模型纵向拆分，每个 GPU 只包含模型的一部分层，数据在一个 GPU 完成运算后，将输出传给下一个 GPU 继续计算。GPipe^[60]对每层的计算设计一个成本计算函数，给定划分的数量 K ，GPipe 会根据成本计算函数将 L 层网络分成 K 组，每组中包含一定连续的层数。在前向计算的过程中，GPipe 会将 batch 划分成 M 份相等的微批次 (micro-batch)，在 M 个计算设备之间进行流水。在反向传播计算期间，每个微批次的梯度是基于用于前向传播计算的模型参数计算的，将来自所有 M 个微批次的梯度累加起来，并应用于更新所有计算设备的模型参数。DAPPLE^[61]是一个同步分布式训练框架，它结合了大型 DNN 模型的数据并行性和流水并行性，主要思想是更早地安排反向传播任务，从而释放用于存储相应前向传播任务产生的激活内存。对比 GPipe 的调度机制，DAPPLE 不是一次性执行所有 M 个

表4 不同并行算法比较

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
数据并行	Parameter Server ^[58]	2013年	中	中	高	中	较高	较高	中
	Ring AllReduce ^[59]	2017年	较高	高	高	较高	高	高	较高
流水线并行	GPipe ^[60]	2019年	中	高	高	低	较高	高	高
	DAPPLE ^[61]	2021年	高	较高	高	中	高	中	高
	PipeDream ^[62]	2019年	高	高	较高	中	较高	较高	较高
张量并行	Megatron-LM ^[63]	2019年	高	中	高	低	较高	中	高
	Colossal AI ^[64]	2021年	高	高	高	高	中	较高	高
	Low-bit Communication ^[65]	2024年	高	中	高	中	中	较高	中

微型批次 (mini-batch)，而是在开始时执行 K 个 mini-batch ($K < M$)，从而释放内存压力，同时到达内存使用的最高点。其次，严格安排一个 mini-batch 的前向传播后，紧接着就是一个反向传播，以保证反向传播可以提前执行。PipeDream^[62]创新性地采用了一前一后 (one forward pass followed by one backward pass, 1F1B) 并行计算模式，该模式通过交替执行前向传播和反向传播来实现高效计算。在 1F1B 机制下，前向计算与反向计算交错进行，这种设计能够有效释放不再需要的中间计算结果，从而优化内存使用效率。

张量并行指的是将模型横向拆分，将模型的每一层拆分开，放到不同的 GPU 上，每一层的计算都需要多个 GPU 合作完成。Megatron-LM^[63]将多头注意力机制中的每个头分配到不同 GPU 上计算，多层感知机 (multi-layer perceptron, MLP) 层则通过列并行和行并行的组合实现高效计算，同时减少同步点和优化通信操作 (如 all-reduce)，降低通信开销。然而，此前的张量并行都是 1 维张量并行，即只划分了模型权重参数而没有划分激活值。为此，Colossal AI^[64]实现了多维张量并行，分别从 2、2.5、3 这 3 种维度进行并行计算。其中 2 维指的是同时沿着权重和激活值维度进行划分。2.5 维并行使用更多的处理器来

减少 2 维并行中的通信，3 维并行则是对模型的矩阵进行了更细粒度的划分。Dong 等^[65]认为随着服务器 LLM 规模不断扩大，张量需要分布在更多设备上，导致通信成本大大增加，因此他们提出低比特通信 (low-bit communication)，利用通信特征中的一致异常值，他们引入了一种量化方法，该方法将平均通信值从 16 位减少到 4.2 位，同时保留了几乎所有的原始性能。

2.2.2 内存访问优化算法

不同内存访问优化算法的对比见表 5。

分块 (tiling) 是一种重要的技术，旨在将大规模计算任务分解为小块，减少内存访问的频繁性和数据移动的开销，从而提高计算效率。FlashAttention^[66]通过分块计算和 I/O 感知的优化策略，显著减少了注意力机制中的内存访问开销。它将注意力矩阵分块加载到高速缓存 (如 GPU 的共享内存) 中，避免了矩阵乘法频繁访问高时延的全局内存，从而在长序列推断中实现了更高的速度和更低的内存占用。LongLoRA^[25]在上下文扩展任务中引入了 LoRA 方法，并结合分块计算技术，降低对硬件资源的需求。它采用切换短期注意力 (shift short attention) 机制，将注意力计算改进为分组计算，并通过分块策略

表 5 不同内存访问优化算法比较

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
分块	FlashAttention ^[66]	2022 年	高	高	高	中	较高	高	高
	LongLoRA ^[25]	2023 年	中	高	高	中	高	高	高
	MLC LLM ^[67]	2023 年	中	高	高	低	较高	高	较高
预取	LazyLLM ^[68]	2024 年	中	高	高	中	较高	较高	高
	CoA ^[69]	2025 年	高	较高	高	中	高	较高	高
	PRESERVE ^[70]	2025 年	高	高	高	中	较高	较高	较高

保障相邻组间的信息共享。MLC LLM^[67]通过分块计算和动态形状优化，减少了对最大输入长度的额外填充需求，从而降低了计算量和内存使用量。它利用 TVM 编译器的优化能力，实现了高效的分块矩阵乘法和内存规划。

预取技术在计算时通过提前加载数据或中间结果，降低内存访问时延，从而提高推断效率。LazyLLM^[68]通过动态选择重要 token 的分块计算方式，加速长文本推断过程。它利用预取技术，提前加载和处理关键 token 的隐含状态，从而减少不必要的计算。LazyLLM 还引入了 Aux Cache 机制，存储被剪枝 token 的隐含状态，以高效恢复这些 token。CoA (chain-of-agents) ^[69]通过多智能体协作，在多个 LLM 之间利用自然语言进行信息汇聚和上下文推理。它利用预取技术，提前加载和处理分块文本，从而减少长文本推断中的计算开销。PRESERVE^[70]提出将预取命令插入给定的计算图中，使用并行流进行集体通信操作。PRESERVE 还会跟踪预取的数据，并在编译时估计 L2 缓存的使用情况，以使总预取数据大小不超过预定义的上限，以防止缓存污染。

2.2.3 计算图优化

各种不同的计算图优化技术对比见表6。

LLM 推断时每个算子都需要读写显存，当这些算子分开执行时，中间结果需要不断存储到显存中，增加了带宽消耗，为此研究者提出了算子融合 (operator fusion) 技术。算子融合的主要思想是将模型中多个连续算子合并成单个算子，这单个算子全部在显存中进行，当计算完成后才一起写回内存。算子融合能够显著减少中间结果的存储和传递，减少内存访问次数，提高硬件的计算利用率。FlashAttention^[66]将整个注意力操作中的算子（键矩阵与查询矩阵相乘，Softmax、注意力权重矩阵与值矩阵相乘等）融合为一个算子，并将键、值、注意力权重矩阵平铺为很多块，从而避免完整数据加载的开销。FlashDecoding^[71]建立在FlashAttention的基础上，因为查询矩阵在推断时仅有一行，对GPU的使用率非常低。因此它将键、值矩阵拆分成更小的块，然后使用FlashAttention并行计算这些小块中的注意力结果，最后计算这些小块计算结果的加权和作为最终的结果。

表6 不同计算图优化算法比较

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
算子融合	FlashAttention ^[66]	2022年	高	高	高	中	较高	高	高
	FlashDecoding ^[71]	2023年	较高	高	高	中	较高	较高	较高
	FlashDecoding++ ^[72]	2023年	高	高	高	中	较高	较高	高
	Atom ^[73]	2024年	中	高	高	低	较高	较高	较高
	RPTQ ^[74]	2023年	中	较高	高	中	较高	较高	较高
动态早退	DeeNERT ^[75]	2020年	中	高	较高	低	较高	较高	高
	Vocabulary Pruning ^[76]	2024年	中	较高	高	中	较高	较高	较高
	EE-LLM ^[77]	2023年	高	高	高	中	高	高	高

FlashDecoding++^[72]进 一 步 优 化 了 FlashDecoding，它给每个小块的注意力计算中 Softmax 函数设置统一的最大值，从而避免了不同分块中 Softmax 函数结果同步带来的额外开销。此外，其他长文本推断优化技术中的方案可能会带来额外计算开销，为此研究者们使用算子优化来减少这些开销。例如，量化操作中需要有解量化的步骤，而这会带来额外的计算量。Atom^[73]将解量化算子和矩阵乘法算子融合，消除了解量化对计算效率的影响。RPTQ^[74]也采用和 AWQ 一样的重排序操作来进行混合精度量化，但是文献[74]指出重排序也会带来计算开销，为此它将重排序算子与 LLM 中的层正则化算子融合。当层正则化计算完成，将要写回显存时，它将写回后的地址改为排序后的地址。

动态早退（dynamic early exit）技术是一种根据输入数据的复杂度动态调整模型计算路径的方法。它不执行完所有的层，而是提前退出计算，以减少冗余计算量、提升推断效率。DeeBERT^[75]通过在 BERT 模型的每一层添加早退模块，动态决定是否在当前层输出结果。早退决策基于当前

层的输出置信度分数，如果置信度超过预设阈值，则提前退出推断过程，从而减少计算量。Vincenti 等^[76]指出现代 LLM 中的词汇量很大，导致早退决策所需的置信度估计在计算上成本高昂，从而降低了效率增益。为此，他们提出 Vocabulary Pruning，在推断时为每个标记动态修剪词汇表。具体来说，词汇表在初始层之一被修剪，然后在其余的前向传播中使用较小的词汇表。EE-LLM^[77]以 Megatron-LM 为基础，实现了各种针对早退的算法创新和性能优化，包括一种通过流水线并行促进早退训练目标的反向传播的轻量级方法、一种利用原始流水线计划中的空闲资源进行与早退层相关的计算的技术，以及两种与 KV Cache 兼容的早退推理方法，用于自回归生成。

2.3 系统优化

系统优化主要包括内存分配优化、异构资源管理以及批处理 3 个方面。各种不同系统优化算法的对比见表 7。

表7 不同系统优化技术比较

分类	论文	年份	复杂度	效率	精度	成本	可扩展性	灵活性	创新性
内存分配优化	S ³ ^[78]	2023 年	中	较高	中	较低	较高	较高	较高
	vLLM ^[79]	2023 年	中	较高	高	低	高	高	高
	LightLLM ^[80]	2024 年	较高	较高	高	较低	较高	较高	较高
异构资源管理	FlexGen ^[51]	2023 年	中	高	较高	低	较高	较高	较高
	PowerInfer ^[81]	2023 年	较高	高	高	较低	高	高	高
	PowerInfer2 ^[82]	2024 年	高	高	高	较低	高	高	高
	FastDecode ^[83]	2024 年	较高	高	较高	较低	较高	较高	较高
批处理	ORCA ^[84]	2022 年	较高	高	中	较低	较高	较高	较高
	Deepspeed-fastgen ^[85]	2024 年	较高	高	中	较低	较高	较高	较高
	BatchLLM ^[86]	2024 年	高	中	高	高	较高	高	高

2.3.1 内存分配优化

当 LLM 推断时，如果输入的上下文长度过长，会使 KV Cache 的显存消耗量很大。此外，LLM 的输出文本长度通常不固定，系统不知道应该分配多少显存，导致显存经常浪费。最早的 LLM 根据每个请求的预设最大长度预先分配存储空间。但是，如果提前结束输出的生成，这种方法同样会导致显存浪费。 S^{31781} 为每个请求预测生成长度的上限，以减少预分配空间的浪费。但是这仍然是静态的显存分配，与操作系统的内存分配一样，静态的分配策略会导致大量的显存碎片出现。为此，vLLM^[79] 模仿操作系统的内存分页操作，当请求来临时，vLLM 为 KV Cache 分配连续的逻辑显存分页，而这些连续的逻辑显存分页被映射到不连续的物理显存分页，从而充分利用显存碎片。此外，它还使用了写入时复制（copy-on-write）机制，即只有需要修改的时候才会复制，从而大大降低显存占用。在 vLLM 的基础上，LightLLM^[80] 使用更细粒度的 KV Cache（将令牌的 KV 缓存作为一个单元来处理，而不是一个块）存储来减少不规则边界所产生的浪费。

2.3.2 异构资源管理

当 GPU 显存资源紧张时，可以考虑将部分数据卸载到异构资源，如资源充足的 CPU 内存中。CPU 的内存容量很大，但是读取速度较慢，因此需要谨慎考虑卸载的数据内容，一般只有一部分数据，且都是一些读取频率较低的“冷”数据。FlexGen^[51] 制定了一个开销模型来确定数据在 CPU 和 GPU 上计算的开销以及在 CPU 与 GPU 之间调度的开销，通过求解这个开销模型来确定数据分配在 CPU 和 GPU

上的比例；同时使用了流水线策略，即下一批数据的加载和前一批数据的存储可以与当前批的计算重叠。文献[81]通过实验研究发现，LLM 中仅有少量神经元（定义为热神经元）在不同输入条件下保持持续激活状态，而其余大部分神经元（定义为冷神经元）的激活状态与具体输入内容相关。基于这一发现，PowerInfer^[81] 采用了一种创新的 GPU-CPU 混合推理架构：将频繁激活的热神经元预先加载至 GPU 以实现快速访问，而将输入依赖型的冷神经元交由 CPU 进行计算处理。PowerInfer2^[82] 在 PowerInfer 的基础上，进一步设计了一个多态神经元引擎，可以适应 LLM 推断的各个阶段的计算策略。此外，它引入了分段神经元缓存和细粒度神经元集群级流水线，有效地减少和隐藏了由不同存储介质之间调度的 I/O 操作引起的开销。FastDecode^[83] 发明了一个序列级负载稳定调度器，以最大限度地减少 CPU 和 GPU 的空转。CPU 上的工作负载与它维护的序列的总长度成正比。为了保持时延稳定，序列按照工作负载控制算法输入系统。短序列和长序列由 CPU 工作单元（CPU worker）同时处理，使序列的总长度保持稳定。

2.3.3 批处理

LLM 在线服务应用常常需要处理来自多个用户的请求，此时需要将多个请求作为一个批次一起推理。然而，每个请求的输出文本长度不同，即使一些请求序列长度很短，它们仍然需要等待输出较长的请求结束后才能返回，这些请求会增加很多响应时延。ORCA^[84] 创新性地采用了连续批处理机制，该技术充分利用服务场景中的请求异步特性，在较短输出请求完成时立即终止处理，转而对新到达请求进行批

处理。在该框架中，每个请求的计算过程由多个迭代组成，每个迭代对应预填充或解码阶段。基于这一设计，研究者进一步提出了迭代级别的跨请求批处理方案。Deepspeed-fastgen^[85]在此基础上引入了分割融合技术，该技术首先将较长的预填充请求在序列维度进行切分，随后与多个短解码请求进行混合批处理。这种创新方法有效平衡了不同迭代间的工作负载，并通过消除新请求等待时间显著降低了系统尾部时延。BatchLLM^[86]在全局范围内明确标识公共前缀，共享相同前缀的请求将被一起调度以最好地重用KV上下文。同时它重新排序请求，并首先调度具有较大解码比例请求，以便更好地将解码令牌与后面的预填充块混合，并应用以内存为中心的令牌批处理来扩大令牌批处理大小，这有助于提高GPU的利用率。最后，BatchLLM使用水平融合优化前缀共享注意内核，以减少尾部效应和内核启动开销。

3 总结与展望

随着LLM技术的高速发展，其能处理的上下文越来越长。然而在LLM推断时如果输入文本很长，可能会导致显存消耗显著增加和推断时延显著提高等一系列问题。如何优化LLM在长文本推断时的效率，成为现在LLM研究领域的一个关键挑战。

现有的技术主要可以分为模型优化、计算优化、系统优化3类。每种优化方法都有不同的侧重点，适用于不同的场景，并且在性能、效率、硬件要求等方面有所不同。模型优化通过设计高效注意力机制、设计高效模型架构以及模型压缩等技术，能够在保持较高推理精度的同时显著减少模型参数数量和计算量，但其开发成本较高，

且过度优化可能导致精度损失和泛化能力下降。计算优化则专注于提升硬件利用率和推理速度，通过并行算法、内存访问优化算法以及计算图优化等技术实现高效计算，但其效果高度依赖于硬件架构，移植到其他平台可能需要重新优化，且性能提升有限。系统优化通过内存分配优化、异构资源管理以及批处理等技术，能够高效调度资源和支持大规模部署，但其实现复杂度高，部署和维护成本较高，且对单次推理的性能提升可能不如模型优化和计算优化显著。

总之，虽然当前的优化技术已经取得了许多进展，但是仍然存在模型精度下降较多、实现复杂度较高、实现过于依赖特定硬件、优化力度不够等问题，目前仍有大量潜在的研究方向值得探索。

首先是稀疏性与动态激活。虽然稀疏计算能够减少计算量，但如何更智能地动态选择稀疏激活路径，进一步减少冗余计算，仍是一个关键问题。利用MoE等架构，可以在不牺牲精度的情况下更高效地进行推断，未来可以探索如何在更多任务和更大规模的模型中应用这种方法。

其次是长程依赖的高效捕捉。现有技术如S4^[39]和Linformer^[23]等通过状态空间模型或低秩近似优化长文本的处理效率，但如何在长程依赖与局部信息之间找到更优的平衡，依然需要更深入的理论与实践研究，特别是在提升推理速度的同时保持生成任务的准确性。

接着是能效优化与硬件适配。尽管当前的低精度量化技术（如GPTQ^[45]、LLM.int8()^[46]）在减少能耗方面表现突出，但如何更好地结合硬件特性，如适应不同GPU、张量处理单元（tensor processing unit, TPU）的计算架构来优化推理，仍有提升空间。硬件加速器专用优化算法的设

计将成为未来模型推理中的一大关键领域。

最后是记忆增强与上下文存储。长文本推断的一个挑战是如何高效管理长程记忆。如 Mamba^[40]曾经采用了一种在推理过程中使用全局记忆模块的方式，但如何将这些模块进一步优化，以适应极长的上下文输入，是未来需要重点研究的方向。

综上所述，LLM 长文本推断要求在现有优化技术上进行不断的创新和改进。未来的研究可以围绕智能动态激活、长程依赖处理、硬件加速和记忆增强技术展开，从而更好地应对未来 LLM 长文本推断带来的挑战，确保推断时的高效性和准确性。

参考文献：

- [1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Proceedings of the 31st International Conference on Neural Information Processing Systems. New York: ACM, 2017: 6000 – 6010.
- [2] DEVLIN J, CHANG M W, LEE K, et al. BERT: pre-training of deep bidirectional transformers for language understanding[EB]. arXiv preprint, 2018, arXiv: 1810.04805.
- [3] RADFORD A, NARASIMHAN K. Improving language understanding by generative pre-training[Z]. 2018.
- [4] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[J]. OpenAI Blog, 2019, 1 (8): 9.
- [5] BROWN T B. Language models are few-shot learners[EB]. arXiv preprint, 2020, arXiv: 2005.14165.
- [6] ACHIAM J, ADLER S, AGARWAL S, et al. Gpt-4 technical report[EB]. arXiv preprint, 2023, arXiv: 2303.08774.
- [7] ELMAN J. Finding structure in time[J]. Cognitive Science, 1990, 14(2): 179–211.
- [8] GRAVES A. Supervised sequence labeling with recurrent neural networks[M]. Heidelberg: Springer, 2012.
- [9] LIU Y H, OTT M, GOYAL N, et al. RoBERTa: a robustly optimized BERT pretraining approach[EB]. arXiv preprint, 2019, arXiv: 1907.11692.
- [10] CHOWDHURY A, NARANG S R, DEVLIN J, et al. PaLM: scaling language modeling with pathways[EB]. arXiv preprint, 2022, arXiv: 2204.02311.
- [11] TOUVRON H, LAVRIL T, IZACARD G, et al. LLaMA: open and efficient foundation language models[EB]. arXiv preprint, 2023, arXiv: 2302.13971.
- [12] TOUVRON H, MARTIN L, STONE K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. [EB]. arXiv preprint, 2023, arXiv: 2307.09288.
- [13] SHAZEER N. Fast transformer decoding: one write-head is all you need[EB]. arXiv preprint, 2019, arXiv: 1911.02150.
- [14] AINSLIE J, LEE-THORP J, DE JONG M, et al. GQA: training generalized multi-query transformer models from multi-head checkpoints[EB]. arXiv preprint, 2023, arXiv: 2305.13245.
- [15] LIU A, LIU J, PAN Z Z, et al. Mini-Cache: KV cache compression in depth dimension for large language models [EB]. arXiv preprint, 2024, arXiv: 2405.14366.
- [16] WU H Y, TU K W. Layer-condensed KV cache for efficient inference of large language models[C]//Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. Stroudsburg: ACL, 2024: 11175–11188.

- [17] BRANDON W, MISHRA M, NRUSIMHA A, et al. Reducing transformer key-value cache size with cross-layer attention[EB]. arXiv preprint, 2024, arXiv: 2405.12981.
- [18] XIAO G X, TIAN Y D, CHEN B D, et al. Efficient streaming language models with attention sinks[EB]. arXiv preprint, 2023, arXiv: 2309.17453.
- [19] BELTAGY I, PETERS M E, COHAN A. Longformer: the long-document transformer[EB]. arXiv preprint, 2020, arXiv: 2004.05150.
- [20] ZHANG Z Y, SHENG Y, ZHOU T Y, et al. H₂O: heavy-hitter oracle for efficient generative inference of large language models[C]//Advances in Neural Information Processing Systems. [S.l.: s. n.], 2024.
- [21] MUNKHDALAI T, FARUQUI M, GOPAL S. Leave No context behind: efficient infinite context transformers with infinite attention[EB]. arXiv preprint, 2024, arXiv: 2404.07143.
- [22] CHOROMANSKI K, LIKHOSHERSTOV V, DOHAN D, et al. Rethinking attention with performers[EB]. arXiv preprint, 2020, arXiv: 2009.14794.
- [23] WANG S N, LI B Z, KHABSA M, et al. Linformer: self-attention with linear complexity[EB]. arXiv preprint, 2020, arXiv: 2006.04768.
- [24] HU E J, SHEN Y, WALLIS P, et al. Lora: low-rank adaptation of large language models[EB]. arXiv preprint, 2021, arXiv: 2106.09685.
- [25] CHEN Y K, QIAN S J, TANG H T, et al. LongLoRA: efficient fine-tuning of long-context large language models[EB]. arXiv preprint, 2023, arXiv: 2309.12307.
- [26] LI S Y, LU H F, WU T Y, et al. CaraServe: CPU-assisted and rank-aware LoRA serving for generative LLM inference[EB]. arXiv preprint, 2024, arXiv: 2401.11240.
- [27] SHAZEER N, MIRHOSEINI A, MAZIARZ K, et al. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer[EB]. arXiv preprint, 2017, arXiv: 1701.06538.
- [28] FEDUS W, ZOPH B, SHAZEER N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity[J]. Journal of Machine Learning Research, 2022, 23(120): 1-39.
- [29] DU N, HUANG Y P, DAI A M, et al. GLaM: efficient scaling of language models with mixture-of-experts[EB]. arXiv preprint, 2021, arXiv: 2112.06905.
- [30] ZHANG Z Y, LIN Y K, LIU Z Y, et al. MoEification: transformer feed-forward layers are mixtures of experts[EB]. arXiv preprint, 2021, arXiv: 2110.01786.
- [31] LEWIS M, BHOSALE S, DETTMERS T, et al. BASE layers: simplifying training of large, sparse models[EB]. arXiv preprint, 2021, arXiv: 2103.16716.
- [32] ZHAO Z Y, DONG M H, ZHANG R Y, et al. FactorLLM: factorizing knowledge *via* mixture of experts for large language models[EB]. arXiv preprint, 2024, arXiv: 2408.11855.
- [33] ZOPH B, LE Q V, MATHUR V, et al. Neural architecture search with reinforcement learning[EB]. arXiv preprint, 2016, arXiv: 1611.01578.
- [34] YIN Y C, CHEN C, SHANG L F, et al. AutoTinyBERT: automatic hyperparameter optimization for efficient pre-trained language models[EB]. arXiv preprint, 2021, arXiv: 2107.13686.

- [35] BENDER G, KINDERMANS P J, ZOPH B, et al. Understanding and simplifying one-shot architecture search[C]//Proceedings of the International Conference on Machine Learning. [S.l.: s.n.], 2018: 550–559.
- [36] SARAH A, SRIDHAR S N, SZANKIN M, et al. LLaMA–NAS: efficient neural architecture search for large language models[EB]. arXiv preprint, 2024, arXiv: 2405.18377.
- [37] KLEIN A, GOLEBIEWSKI J, MA X C, et al. Structural pruning of pre-trained language models *via* neural architecture search[EB]. arXiv preprint, 2024, arXiv: 2405.02267.
- [38] JAVAHERIPI M, DE ROSA G, MUKHERJEE S, et al. LiteTransformerSearch: training-free neural architecture search for efficient language models[J]. Advances in Neural Information Processing Systems, 2022, 35: 24254–24267.
- [39] GU A, GOEL K, RÉ C. Efficiently modeling long sequences with structured state spaces[EB]. arXiv preprint, 2021, arXiv: 2111.00396.
- [40] GU A, DAO T. Mamba: linear-time sequence modeling with selective state spaces[EB]. arXiv preprint, 2023, arXiv: 2312.00752.
- [41] PENG B, ALCAIDE E, ANTHONY Q, et al. RWKV: reinventing RNNs for the transformer era[EB]. arXiv preprint, 2023, arXiv: 2305.13048.
- [42] BECK M, PÖPPEL K, SPANRING M, et al. xLSTM: extended long short-term memory[EB]. arXiv preprint, 2024, arXiv: 2405.04517.
- [43] FENG L, TUNG F, AHMED M O, et al. Were RNNs all we needed? [EB]. arXiv preprint, 2024, arXiv: 2410.01201.
- [44] HAN S, MAO H Z, DALLY W J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding[EB]. arXiv preprint, 2015, arXiv: 1510.00149.
- [45] FRANTAR E, ASHKBOOS S, HOEFLE T, et al. GPTQ: accurate post-training quantization for generative pre-trained transformers[EB]. arXiv preprint, 2022, arXiv: 2210.17323.
- [46] DETTMERS T, LEWIS M, BELKADA Y, et al. LLM.int8(): 8-bit matrix multiplication for transformers at scale[EB]. arXiv preprint, 2022, arXiv: 2208.07339.
- [47] KIM S, HOOPER C, GHOLAMI A, et al. SqueezeLLM: dense-and-sparse quantization[EB]. arXiv preprint, 2023, arXiv: 2306.07629.
- [48] LIN J, TANG J M, TANG H T, et al. AWQ: activation aware weight quantization for LLM compression and acceleration[EB]. arXiv preprint, 2023, arXiv: 2306.00978.
- [49] LIU Z R, YUAN J Y, JIN H Y, et al. KIVI: a tuning-free asymmetric 2bit quantization for KV cache[EB]. arXiv preprint, 2024, arXiv: 2402.02750.
- [50] YANG J Y, KIM B, BAE J, et al. No token left behind: reliable KV cache compression *via* importance-aware mixed precision quantization[EB]. arXiv preprint, 2024, arXiv: 2402.18096.
- [51] SHENG Y, ZHENG L M, YUAN B H, et al. Flexgen: high-throughput generative inference of large language models with a single GPU[C]//Proceedings of the International Conference on Machine Learning. [S.l.: s.n.], 2023: 31094–31116.
- [52] HINTON G, VINYALS O, DEAN J. Distilling the knowledge in a neural network

- [EB]. arXiv preprint, 2015, arXiv: 1503.02531.
- [53] GU Y X, DONG L, WEI F R, et al. Mini-LLM: knowledge distillation of large language models[EB]. arXiv preprint, 2023, arXiv: 2306.08543.
- [54] AGARWAL R, VIEILLARD N, STAŃCZYK P, et al. GKD: generalized knowledge distillation for autoregressive sequence models[EB]. arXiv preprint, 2023, arXiv: 2306.13649.
- [55] ZHAO J C, ZHAO W L, DROZDOV A, et al. Multistage collaborative knowledge distillation from a large language model for semi-supervised sequence generation [EB]. arXiv preprint, 2023, arXiv: 2311.08640.
- [56] WU M H, WAHEED A, ZHANG C Y, et al. LaMini-LM: a diverse herd of distilled models from large-scale instructions[EB]. arXiv preprint, 2023, arXiv: 2304.14402.
- [57] YANG R M, WU T Q, WANG J H, et al. LLM-NEO: parameter efficient knowledge distillation for large language models[EB]. arXiv preprint, 2024, arXiv: 2411.06839.
- [58] LI M, ZHOU L, YANG Z C, et al. Parameter server for distributed machine learning[C]//Big Learning NIPS Workshop. [S.l.: s.n.], 2021.
- [59] GIBIANSKY A, DIAMOS G. baidu-allreduce [EB]. GitHub, 2017.
- [60] HUANG Y P, CHENG Y L, BAPNA A, et al. GPipe: efficient training of giant neural networks using pipeline parallelism[EB]. arXiv preprint, 2018, arXiv: 1811.06965.
- [61] FAN S Q, RONG Y, MENG C, et al. DAPPLE: a pipelined data parallel approach for training large models[C]//Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM, 2021: 431-445.
- [62] NARAYANAN D, HARLAP A, PHANISHAYEE A, et al. PipeDream: generalized pipeline parallelism for DNN training[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. New York: ACM, 2019: 1-15.
- [63] SHOEYBI M, PATWARY M, PURI R, et al. Megatron-LM: training multi-billion parameter language models using model parallelism[EB]. arXiv preprint, 2019, arXiv: 1909.08053.
- [64] LI S G, XUE F Z, BARANWAL C, et al. Sequence parallelism: long sequence training from system perspective[EB]. arXiv preprint, 2021, arXiv: 2105.13120.
- [65] DONG H, JOHNSON T, CHO M, et al. Towards low-bit communication for tensor parallel LLM inference [EB]. arXiv preprint, 2024, arXiv: 2411.07942.
- [66] DAO T, FU D, ERMON S, et al. Flash attention: fast and memory-efficient exact attention with io-awareness[J]. Advances in Neural Information Processing Systems, 2022, 35: 16344-16359.
- [67] FENG S Y, HOU B H, JIN H Y, et al. TensorIR: an abstraction for automatic tensorized program optimization[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. New York: ACM, 2023: 804-817.
- [68] FU Q C, CHO M, MERTH T, et al. LazyLLM: dynamic token pruning for efficient long context LLM inference [EB].

- arXiv preprint, 2024, arXiv: 2407.14057.
- [69] ZHANG Y S, SUN R X, CHEN Y F, et al. Chain of agents: large language models collaborating on long-context tasks[J]. *Advances in Neural Information Processing Systems*, 2025, 37: 132208–132237.
- [70] YÜZÜGÜLER A C, ZHUANG J W, CAVIGELLI L. PRESERVE: prefetching model weights and KV-cache in distributed LLM serving[EB]. arXiv preprint, 2025, arXiv: 2501.08192.
- [71] DAO T, HAZIZA D, MASSA F, et al. Flash-decoding for long-context inference[Z]. 2023.
- [72] HONG K, DAI G H, XU J M, et al. FlashDecoding++: faster large language model inference on GPUs[EB]. arXiv preprint, 2023, arXiv: 2311.01282.
- [73] ZHAO Y L, LIN C Y, ZHU K, et al. Atom: low-bit quantization for efficient and accurate LLM serving[J]. *Proceedings of Machine Learning and Systems*, 2024, 6: 196–209.
- [74] YUAN Z H, NIU L, LIU J W, et al. RPTQ: reorder-based post-training quantization for large language models[EB]. arXiv preprint, 2023, arXiv: 2304.01089.
- [75] XIN J, TANG R, LEE J, et al. DeBERT: dynamic early exiting for accelerating BERT inference[EB]. arXiv preprint, 2020, arXiv: 2004.12993.
- [76] VINCENTI J, ABDEL SADEK K, VELJA J, et al. Dynamic vocabulary pruning in early-exit LLMs[EB]. arXiv preprint, 2024, arXiv: 2410.18952.
- [77] CHEN Y X, PAN X C, LI Y L, et al. EE-LLM: large-scale training and inference of early-exit large language models with 3D parallelism[EB]. arXiv preprint, 2023, arXiv: 2312.04916.
- [78] JIN Y, WU C F, BROOKS D, et al. S³: increasing GPU utilization during generative inference for higher throughput[J]. *Advances in Neural Information Processing Systems*, 2023, 36: 18015–18027.
- [79] KWON W, LI Z H, ZHUANG S Y, et al. Efficient memory management for large language model serving with PagedAttention[C]//*Proceedings of the 29th Symposium on Operating Systems Principles*. New York: ACM, 2023: 611–626.
- [80] ModelTC. LightLLM[Z]. 2024.
- [81] SONG Y X, MI Z Y, XIE H T, et al. PowerInfer: fast large language model serving with a consumer-grade GPU[EB]. arXiv preprint, 2023, arXiv: 2312.12456.
- [82] XUE Z L, SONG Y X, MI Z Y, et al. PowerInfer-2: fast large language model inference on a smartphone[EB]. arXiv preprint, 2024, arXiv: 2406.06282.
- [83] HE J A, ZHAI J D. FastDecode: high-throughput GPU-efficient LLM serving using heterogeneous pipelines[EB]. arXiv preprint, 2024, arXiv: 2403.11421.
- [84] YU G I, JEONG J S, KIM G W, et al. Orca: a distributed serving system for {Transformer-Based} generative models[C]//*Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation*. [S. l.: s. n.], 2022: 521–538.
- [85] HOLMES C, TANAKA M, WYATT M, et al. DeepSpeed-FastGen: high-throughput text generation for LLMs via MII and DeepSpeed-inference[EB]. arXiv preprint, 2024, arXiv: 2401.08671.
- [86] ZHENG Z, JI X, FANG T S, et al. BatchLLM: optimizing large batched LLM inference with global prefix sharing and throughput-oriented token batching[EB]. arXiv preprint, 2024, arXiv: 2412.03594.

作者简介



陶伟（1999–），男，华中科技大学博士生，主要研究方向为模型量化、云边协同、大模型时序数据应用、大模型长文本推断优化等。在体系结构方向国际顶级会议发表论文2篇，在人工智能方向国际顶级会议发表论文1篇。



王健宗（1983–），男，博士，平安科技（深圳）有限公司副总工程师，资深人工智能总监，联邦学习技术部总经理，智能金融前沿技术研究院院长。美国佛罗里达大学人工智能博士后，美国莱斯大学和华中科技大学联合培养博士，中国计算机学会资深会员，中国计算机学会大数据专家委员会委员，中国自动化学会联邦数据和联邦智能专业委员会副主任。主要研究方向为大模型、联邦学习和深度学习等。



张旭龙（1988–），男，博士，平安科技（深圳）有限公司，高级算法研究员，复旦大学计算机理学博士，2023年入选上海市东方英才计划青年项目。兼任深圳清华大学研究院及中国科学技术大学先进技术研究院校外导师，中国自动化学会联邦数据与联邦智能专业委员会委员。目前是IEEE、中国自动化学会以及中国计算机学会会员。主要研究方向为大模型、具身智能、跨模态智能计算以及模型上下文协议（MCP）等。发表学术论文80余篇，申请国家发明专利授权20余项。



瞿晓阳（1988–），男，博士，平安科技（深圳）有限公司前沿机器学习算法分组负责人，清华大学深圳国际研究生院校外导师，中国科技大学先进技术研究院校外导师，中佛罗里达大学访问学者，从事机器学习、大数据、体系结构方面的研究工作，在语音语义分析、自动化机器学习、零样本和小样本学习、高性能计算与存储等方面经验丰富。在体系结构方向（如INFOCOM、DAC、IPDPS、TPDS）和人工智能方向（如NeurIPS、IJCAI、ICASSP、Interspeech）等国际会议和期刊发表过近50篇文章，其中1篇论文荣获会议最佳学生论文奖提名；担任多个国际期刊的评委，已授权专利70篇，已出版专著2本。

收稿日期: 2025-03-19
通信作者: 王健宗, jzwang@188.com; 瞿晓阳, quxiaoy@gmail.com
基金项目: 广东省重点领域研发计划“新一代人工智能”重大专项(No.2021B0101400003)
Foundation Item: The Key Research and Development Program of Guangdong Province (No.2021B0101400003)